Projekt Mastermind

Ausgangslage

Eine Programmiersprache erlernt man am besten, indem man selber möglichst viel programmiert. Als Übungsfeld dient uns das Spiel Mastermind, welches wir als konsolenbasiertes Spiel realisieren wollen. Dabei werden wir in der ersten Phase mit einer Klasse mit mehreren Methoden arbeiten (eigentlich der prozedurale Ansatz). In der zweiten Phase werden wir dann das Problem objektorientiert angehen und verschiedene Klassen für die diversen Teilprobleme erstellen, also ein objektorientiertes Design durchführen.

Vorgehen

- Arbeiten Sie zu zweit, wobei Sie aber Ihren eigenen Code in Ihrer eigenen Lernumgebung erstellen.
- Wenn Sie alleine nicht weiter kommen
  - suchen Sie Hilfe im Team
  - konsultieren Sie die Theorie und die Ressourcen
  - lesen Sie im Buch
  - recherchieren Sie im Internet
- Sie werden in den Arbeitsblättern immer wieder auf Arbeitsaufträge stossen, welche Teilprobleme des Spiels Mastermind zum Inhalt haben.

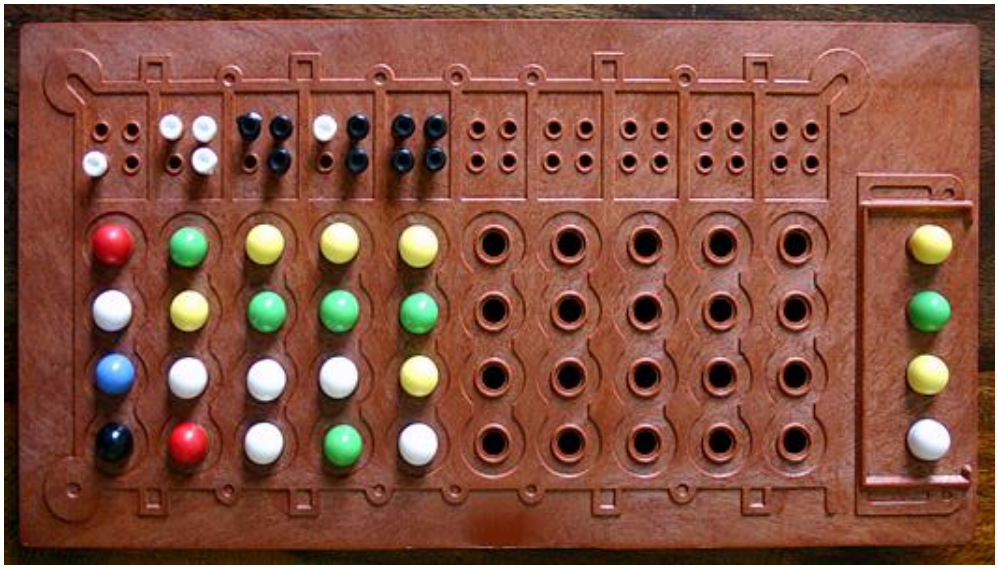
Anforderungen

In diesem Abschnitt werden die Minimalanforderungen an das zu erstellende Spiel präsentiert.

Selbstverständlich dürfen Sie diese Anforderungen ausbauen oder allenfalls anpassen, solange die Idee des Spiels nicht verloren geht. Wenn Sie dies tun, halten Sie die ergänzenden oder angepassten Anforderungen schriftlich fest.

Minimalanforderungen

Mastermind ist ein Spiel für zwei Personen, einen Codierer und einen Rater. Sie finden eine Beschreibung des Originalspiels unter https://de.wikipedia.org/wiki/Mastermind_(Spiel). Sie können im Internet auch lauffähige Masterminds finden, teilweise sogar in Java.



In unserem Fall wird der Codierer durch den Computer realisiert und der Rater durch den Spieler.

Folgender Spielablauf soll realisiert werden:

1. Am Anfang bestimmt der Computer einen zufälligen, vierstelligen Geheimcode aus sechs zur Verfügung stehenden Farben.
2. Nun wird der Spieler aufgefordert den Geheimcode zu erraten und einen eigenen Code einzugeben.
3. Der Computer vergleicht den eingegebenen Code mit dem Geheimcode und zeigt dem Spieler an
    a. wie viele Stifte des eingegebenen Codes korrekte Farbe und Platz haben und
    b. wie viele Stifte zwar die korrekte Farbe, aber nicht den korrekten Platz haben.
4. Wenn der Spieler den Geheimcode erfolgreich aufgedeckt hat ist das Spiel zu Ende. Andernfalls wird der Spieler erneut aufgefordert einen Code einzugeben.

Da auf der Konsole keinerlei grafische Ausgabe möglich ist, sollen die Farben durch einzelne Buchstaben dargestellt werden:

r	Rot
b	Blau
g	Grün
y	Gelb
w	Weiss
s	Schwarz

Die Eingabe des Spielers soll jeweils auf einer neuen Zeile eingelesen werden. Sie soll aus einer Folge von vier Buchstaben ohne Zwischenraum bestehen und wird mit der Return-Taste abgeschlossen.

Das Programm soll die Benutzereingabe auf Korrektheit überprüfen (validieren) und im Falle einer fehlerhaften Eingabe eine entsprechende Meldung ausgeben und zur Wiederholung der Eingabe auffordern.

Bei einer gültigen Eingabe soll auf einer neuen Zeile die Bewertung der Eingabe ausgegeben werden (Schritt 3 in obigem Ablauf).

Die Ausgaben des Programms sollen so gestaltet werden, dass dem Benutzer jederzeit klar ist, was von ihm erwartet wird (Benutzerfreundlichkeit).

Weitere Schritte

Die meisten der folgenden Schritte werden Sie im weiteren Verlauf des Moduls durchführen. Grundsätzlich geht es dabei um folgendes:

1. Analysieren der Anforderungen
   Hier geht es darum festzustellen, ob die Anforderungen verständlich, vollständig und einander nicht widersprechend sind. Andernfalls müssen weitere Abklärungen getroffen werden.

2. Entwurf einer Lösung
   Jetzt müssen Sie verschiedene Dinge festlegen:

   a. Datenstrukturen: Wie wollen Sie im Programm die Werte von Farben, oder dem Geheimcode speichern? Es geht also um Datentypen und Variablen.

   b. Zerlegung des Gesamtproblems in Teilprobleme. Einzelne Teilprobleme müssen eventuell ihrerseits in noch kleinere Teilprobleme zerlegt werden.

      Beispiel:
      - Bestimmen eines zufälligen Geheimcodes
      - Ausgeben der Eingabeaufforderung
      - Einlesen der Benutzereingabe
      - …

   c. Ablaufstrukturen zur Lösung der einzelnen Teilprobleme (Struktogramme, Ablaufdiagramme, etc.)

   d. Texte für die Benutzerschnittstelle

3. Umsetzung der "prozeduralen" Lösung
   Jetzt giessen Sie die Lösung für jedes Teilproblem in eine Java-Methode. Übergeordnete Methoden rufen dabei untergeordnete Methoden auf. Zuoberst steht die main-Methode.

4. Entwurf einer Lösung
   Nun geht es darum Klassen und Beziehungen zwischen Klassen zu finden. Dabei ist es wichtig, dass in Klassen, wie man so schön sagt, nicht Kraut und Rüben gemischt sind. Eine gute Klasse erfüllt deshalb die folgenden Kriterien:

   a. Sie kapselt Daten und Methoden, die eng miteinander zu tun haben.
      Beispiel:
      Eine Klasse Code enthält die Information über den Zustand des Codes (Farben, Positionen, …) und sie bietet Methoden zum verändern dieses Codes an (generieren, …).

   b. Sie repräsentiert ein einziges Konzept und erfüllt eine einzige Aufgabe
      Beispiel:
      Wiederum die Klasse Code: Sie enthält nur Informationen über den Zustand der Codes und bietet nur Methoden zum Verändern desselben an.

   c. Sie delegiert Aufgaben, welche sie nicht selber lösen kann, an andere Klassen, mit welchen sie in Beziehung steht. Dabei gibt sie eigene Informationen weiter.

5. Umsetzung der objektorientierten Lösung
   Jetzt giessen Sie Ihren Klassenentwurf in Java-Klassen. Dabei überprüfen Sie laufend, ob obige Kriterien für gute Klassen eingehalten sind. Allenfalls müssen Sie weitere Klassen einführen oder Ihren Entwurf anpassen.

6. Test der Lösung
   Getestet wird auf zwei Ebenen:
   a. Unit-Tests für einzelne Klassen
   b. Systemtest für das gesamt Programm. Dieser kann in unserem kleinen Beispiel von Hand durchgeführt werden

7. Dokumentation der Klassen und Methoden.
   Wenn Sie die Kriterien für gute Klassen eingehalten haben, dann erschliesst sich der Zweck einer Klasse und ihrer Methoden direkt aus dem Code (vorausgesetzt Sie messen auch sonst der Lesbarkeit Ihres Codes das nötige Gewicht zu).

   Je nach Situation macht es aber auch Sinn, die Aufgabe einer Klasse und ihrer Methoden zusätzlich mit JavaDoc-Kommentaren zu dokumentieren. Dies ist vor allem bei jenen Klassen sinnvoll, welche in andern Anwendungen wieder verwendet werden können (zum Beispiel APIs - Application Programming Interface bzw. Anwendungsprogrammierschnittstelle - ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird).